EMGT 835 FIELD PROJECT:
Implementing Agile Software Development

for Small – Medium Business

By

Galen Blakeman

Master of Science

The University of Kansas

Fall Semester 2008

An EMGT Field Project report submitted to the Engineering Management Program and the Faculty of the Graduate School of The University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Herb Tuttle
Date
Committee Chair

Dr. Robert Zerwekh
Date
Committee Member

Scott Woodward
Date
Committee Member

Acknowledgments

I want to thank my committee members: Herb Tuttle, Dr Robert Zerwekh, and Scott Woodward. Thank you for giving me your valuable time to guide and assist me on my field project.

I would like to thank my two sisters Annette and Michele and their families for all their love and support. I would especially like to thank my brother-in-law Scott Myers for his editing skills.

Executive Summary

Software projects have a long history of delivering projects over budget, behind schedule, and not meeting expectations. Software development teams have typically tried to follow the "waterfall" or building paradigm of "define, design, and develop" in a proscribed fashion. The problem with this approach is it provides no mechanism for innovation or evolution of the software. In reality users often don't really know what they want in software until they see it in action and gain new insights on how they would like it to work.

The contrasting approach is "iterative development" where software is delivered as a series of working features. The family of development processes surrounding iterative development is referred to as "Agile Methodologies" and is characterized as being adaptable to change (Highsmith 2004). This approach embraces the uncertainty surrounding the requirements by measuring the project on vision, cost and schedule rather than scope, cost and schedule. The overall idea with this approach is to turn development into a collaborative process with the customer and illicit feedback early and often. This approach, in turn, allows development to adapt and evolve with change.

The challenge is applying these methodologies to small business environment. In the small business environment, budgets vary from very small to medium in scope, typically on the order of one week to three months. Projects also vary from completely new domain to those that just modify existing features. It is not realistic to follow the same process for these widely varied scenarios. It is also import that costs associated with documenting and managing project must scale with the nature of the project.

The author, through research and experience as a small business developer with Blue Ocean Consulting, lays out an approach that breaks the Agile methodology into phases. These phases provide a framework for skipping aspects of the process that are not needed in certain scenarios.

Projects that fall within a completely new domain would go into a discovery process to define vision, high level features, ballpark of investment, and data sheet. Projects that fall into an existing known domain but have completely new features would skip discovery and start with the analysis phase. Projects that are updates or expansions in the scope of existing features in an existing domain would jump right to the innovation phase.

Table of Contents

1. Introduction

1.1. Overview

Software projects have a long history of failure. These failures cover outright cancellations, budget overruns, schedule overruns, and missing features. The cost of these failures is estimated in the billions of dollars each year. To combat this history of failure, the software industry is slowing changing its methodologies from the traditional assembly-line approach to an agile approach that is adaptive to customer and business needs.

1.2. Blue Ocean Consulting

Blue Ocean Consulting is a small business that targets smaller scale custom software projects that typically fall in the 1 week to 3 months time frame. Its customers typically have limited or no experience with software development. The small business nature of the projects translates to an environment that requires staff to work on multiple projects simultaneously and to fill multiple roles.

1.3. Challenges

Blue Ocean Consulting is unable to provide the full Agile methodology for all the small business projects it handles. In the small business environment it often works on projects where the project is within a known domain or an adaptation of existing

software. In the scenario where there is a known domain it is not feasible for a consultant to justify spending time and money on working through a project vision. In the scenario where it is an adaption of existing software, it is not justifiable to spend time and money on logical design and modeling. However, any software project can benefit by applying Agile to the innovation phase. It is with these challenges in mind that Blue Ocean Consulting has adapted the Agile methodology to small business software consulting.

2. Literature Review

The Agile software development process is a relatively new approach (2001), so there is a limited amount of literature on it. However, it should be pointed out, the process does borrow heavily from the "Lean" manufacturing concepts, specifically the Toyota Production System. During literature review author found that the books and journals agreed on the "Agile" manifesto, but differed on specific application of the Agile process. I didn't find anyone writing books in favor of traditional "waterfall" approaches over Agile.

Books

1. Highsmith, J. A. (2004). Agile project management : creating innovative products. Boston, Addison-Wesley.

The author of the book discusses software development as an innovative process versus assembly line process. The author argues that the businesses need to change their mindset from prescriptive development to adaptive development. In adaptive development, the organization must start with the product vision and document the features to support it at a high level, but they do not detail out requirements. The details are flushed out using iterative feature-based delivery. Underlying relationships, issues, and complexities are only unveiled during actual product development. The development iterations allow the customers to provide feedback on tradeoffs throughout the development cycle of the project, not just at the end. It also allows customers to change or to add features as new insights are gained through process, which increases the value of the project.

2. Anderson, D. J. (2004). Agile management for software engineering : applying the theory of constraints for business results. Upper Saddle River, NJ, Prentice Hall Professional Technical Reference.

The author of this book argues that to fully apply "Agile" processes an organization must utilize production and financial metrics that support it. The author takes his own experience at SprintPCS and argues that traditional cost-based accounting based on effort metrics like lines of code written or the man hour effort expended is dangerous and misleading. The author argues effort-based metrics do not work because software development is non-linear, accurately estimating non-linear activities is impossible, and productivity between individual developers can vary widely. The author argues that throughput accounting is the ideal Agile management approach for software development companies. Throughput accounting measures the value of functions delivered to client and then divides this by operating expense to arrive at the average cost per function (AVPF).

3. Cohn, M. (2004). User stories applied : for agile software development. Boston, Addison-Wesley.

The author of this book argues that the problems with software development are communications related and outlines several specific problems. The first problem he covers is the problem where communications is dominated by either developer or customer. If the communications are dominated by the developer, the user stories end up heavy with technical jargon that does not reflect business language that translates to

valuable features. If communications are dominated by the customer, the software design can specify features without consideration of technical tradeoffs and cannot be completed within budget or timelines. The second problem he outlines is the attempt to predict software design up front. He argues the design of software requires an ongoing dialog where users can see working software in early stages and then form/change their opinions on the implementation of the features. To eliminate these communication problems, the author outlines an approach for the customer and developer team to jointly produce user stories instead of detailed requirements. These user stories, by definition, are action oriented, express value to the user, and are written in business language. The user stories should be short enough to fit on a note card and shouldn't cover implementation details because these cannot be truly known up front.

4. Coplien, J. O. and N. Harrison (2005). Organizational patterns of agile software development. Upper Saddle River, NJ, Pearson Prentice Hall.

This book covers the common elements or "patterns" of agile software organizations. The key patterns outlined for agile development are: "Community of Trust", "Few Roles", "Shared Clear Vision", and "Early and Regular Delivery". "Community of Trust" pattern outlines that the software development process requires human interactions, and developers must establish trust in order to have effective communications. The "Few Roles" pattern refers to eliminating project overhead and latency by reducing the number of team member roles to the level needed to deliver value to customer. The "Shared Clear Vision" pattern outlines that a lack of a clear vision can lead to indecision and contrary opinions, so it is important to establish a statement of purpose for the project up front. The "Early and Regular Delivery" pattern refers to

gaining insight into what did not know early in the project through regular delivery of working code.

5. Larman, C. (2004). Agile and iterative development : a manager's guide. Boston, Addison-Wesley.

The principal argument the author puts forward is that software is inventive product development where the customer wants something custom. Software needs to be custom because the customer wants features tailored to business requirements. The author argues that inventive product development runs in direct contrast to traditional approaches that tried to apply predictive manufacturing techniques to software development. In line with this argument, he states that the best approach to inventive product development is the Agile process because it embraces flexibility and maneuverability. In turn this agility gives businesses a competitive edge.

6. Poppendieck, M. and T. D. Poppendieck (2003). Lean Software Development: An Agile Toolkit. Upper Saddle River, NJ, Addison-Wesley.

This book takes key concepts from lean development and applies them to software development improvement. Lean development references were mainly from Japanese auto manufacturing companies Honda and Toyota and the contrast with American companies like GM and Ford. American companies use highly rigid and documented product development plans, where the Japanese companies Honda and Toyota put emphasis on rapid development and delaying design decisions to later in development cycle. The author shows that Honda and Toyota require half the design time and a third of the development time in comparison to American manufactures. Using the

result of the Japanese lean approach, the author put forward that when applied to software development the Agile toolkit can have even greater results.

Articles

*1. Augustine, S., B. Payne, et al. (2005). "Agile project management: Steering from the edges." Communications of the ACM **48**(12): 85-89.*

The premise of this article is that we are operating in a new "Internet Age" global economy that is increasingly volatile and complex. Traditional formal software development methodologies just cannot adapt and work in this environment. The correct approach to volatile and complex software development is to apply Agile methodologies and "steer from the edges". This Agile approach embraces product adaptability through rapid iterative delivery, flexibility, and working code (Abrahamsson 2003). This approach also drives down decision making. The leaders of the project set forth project vision and high level features, but detail design decisions are pushed down and distributed to developers.

2. Barry Boehm, R. T. (2005). "Management Challenges to Implementing Agile Processes in Traditional Development Organizations." IEEE Software(September/October 2005).

One of the most difficult aspects of implementing an Agile process is the litany of perceived and real barriers that organizations that have used traditional processes put in place; this, of course, is the major premise of this paper. In March 2004, the University of Southern California Center for Software Engineering (USC-CSE) Affiliates Annual Research Review held the fourth in a series of annual workshops to identify as many of these barriers as possible. These barriers were broken into three areas, namely non problems, problems only in terms of size or scope, and significant issues. The non

problems covered the perception that Agile is a fad which are unmanaged and are of inadequate quality. The problems in terms of size and scope covered lack of stakeholder sign-off requirements, lack of planning documentation, does not meet process standards (IEEE, DOD, EIA), and is designing for the battle, not the war. The significant issues were requirements for intense customer communications, difficulties with resource planning, contractual issues, interfacing with multiple systems, and cost estimation. In response to these barriers, the authors outline some basic rules, as follows: Define the team roles within Agile process. Create architectures and organizational structures that support agile process. Realign or redefine traditional project milestones to fit with iterative approach. Apply throughput accounting rather than cost accounting. Update contract structures to target cost or target schedule, so there is shared risk/benefits between customer and producer.

Websites

1. Charette, R. N. (2005). "Why Software Fails." IEEE Spectrum Online Retrieved 2/14/2007, 2007, from http://www.spectrum.ieee.org/sep05/1685.

This article outlines how the majority of software projects fail. It sets forward that between 5-15% are cancelled before or shortly after delivery, and the majority will be delivered late, over budget, and require massive reworking. Software projects fail because of poor project goals, inadequate resources, changing requirements, poor progress reporting, poor communications, and an inability to see complexity up front.

2. Marasco, J. (2006). "Software development productivity and project success rates: Are we attacking the right problem?" IBM developerWorks Retrieved 2/28/2007, 2007, from http://www-128.ibm.com/developerworks/rational/library/feb06/marasco/index.html.

This article outlines the staggering number of software failures, the reasons for the failures, and the different approaches taken. The author pulls from the Standish Group the following software project success rates: 1994: 16%, 2001: 28%, 2003: 31%. As the numbers show, success rates have improved over the last decade, but the failure rate is still staggering. The reason the author makes for this continued failure is that while development and testing tools continually improve they are not keeping pace with changing requirements. In fact, the applications we develop are becoming so rich with features and complexity it is impossible to fully understand their design up front. The author, in conclusion, endorses Agile iterative development due to its light-weight documentation and adaptability to changing requirements.

3. The "Agile" Methodology

3.1. Manifesto

Individuals and interactions	over processes and tools
Working software	over comprehensive documentation
Customer collaboration	over contract negotiation
Responding to change	over following a plan
That is, while there is value in the items on the right, we value the items on the left more.	

Table 1 http://www.agilemanifesto.org/ (Accessed 9/8/2008) (Kent Beck 2001)

3.1.1. Individuals and interactions over processes and tools

Ultimately, unique, talented, and skilled individuals – individually and collectively – build software products and services (Highsmith 2004). The point is that the process and tools do not produce the results. It is the people with the right communication and technical skills that deliver results. The Agile process pushes down detailed decisions to a dialog between developers and customers, which in turn results in product adaptability to increase the success rate for software.

3.1.2. Working software over comprehensive documentation

Software requirements is a communication problem (Cohn 2004) and is inherently at the heart of the reason the majority of software projects fail. The Agile process takes

the approach that it is better to deliver early and often versus documenting every detail of a potential system up front because customers gain better insight with working code. This process also places the emphasis on delivering value to the customer as working code instead of investing resources in comprehensive documentation.

3.1.3. Customer collaboration over contract negotiation

If the software project is tied to a specific design plan that is contractually obligated, there is a tendency for communications with a client to turn into a deluge of change orders and arguments over the exact meaning of a detailed requirement specification. Agile takes a different approach in that it expects the project to evolve during implementation. Under the Agile approach, the project vision and high level features are outlined, and the implementation details are left to ongoing customer collaboration. This ongoing collaboration, through transparency and constant delivery, builds the trust the contract is trying to replace.

3.1.4. Responding to change over following a plan

The key concept here is that software development is an innovation and does not work well with traditional prescriptive approaches. The problem with the prescriptive approach is customers are often not sure what they want, have difficulty stating what they want, and change their mind (Larman 2004). This problem is amplified in the "Internet Age" where the global economy is increasingly volatile and complex (Augustine et al. 2005). The Agile approach embraces product adaptability through rapid iterative delivery, flexibility, and working code (Abrahamsson 2003).

3.2. Benefits

Agile is a software development method that uses collaborative and iterative processes to respond to changing user requirements and delivers high quality software in a cost effective and timely manner.

3.2.1. Collaborative and iterative processes

The Agile project starts by defining the project vision and the supporting user stories (features) at a high level. User stories, by definition, express value to the user and are written in business language. User stories also provide an estimation framework in terms of story points based on effort and complexity. The use of story points reflects the uncertainty of estimates due to the innovative nature of software development, but they still allow for prioritization and budgeting. Based on the prioritization of features, a collaborative iteration plan is developed to deliver working code early and often. Through this combination of transparency and constant delivery, trust is built between development team and stakeholders.

3.2.2. Responding to changing user requirements

Research has shown that typically 25-40% of project defects are related to requirements. In another study, requirements were cited as the largest contributing factor for project failure 80% of the time. The problem relates to the difficulty in defining requirement details up front. Requirement details are difficult to define up front because requirements evolve with changing user needs and market conditions. The requirements also evolve because underlying relationships, issues, and complexities are only unveiled

during actual product development. For this reason, the Agile process delivers working software in two - four week iterations. These iterations give the opportunity to visualize the project early on, which allows new insight and the ability to provide feedback early in the process. The ability to respond to change greatly increases the likelihood of the project meeting stakeholder expectations and being considered a success.

3.2.3. Delivering high quality software

The iterative process delivers working code every few weeks. This iterative process inherently leads to constant feedback and testing. The constant feedback allows the software product to be adaptable and evolve as insights are gained by seeing the software in action. The constant testing allows developers to identify and fix technical issues early before they impact other aspects of the software project. This product adaptability and fixing of technical issues early in the process translates to high quality software.

3.2.4. Cost effective and timely delivery

Agile focuses on delivering working software over creating detailed documentation. While there is value in documenting project needs it is easy to waste time and money trying to flush out the details for a software product, especially knowing that the project details are going to change as insights are gained and user needs evolve. By delivering working software, the Agile process puts the project resources into delivering value to the customer and incorporating changes early when it is most cost effective to do so.

3.3. Agile applied to Blue Ocean Consulting

3.3.1. Challenges

Blue Ocean Consulting is a small business that targets smaller scale custom software projects that typically fall in the one week – three month time frame. These projects can also fall into three typical categories: new domain, new features for an existing domain, and updates to existing features in an existing domain. As a small business, the company has the additional challenge that staff works on multiple projects at a time and fills multiple roles such as development and support.

3.3.2. Phases

To meet challenges outlined above, Blue Ocean Consulting is adopting an Agile methodology that breaks down into phases that tie into the categories of projects it typically works on. These phases are discovery, analysis, and innovation. Projects that are a completely new domain, for example a new application or whole new set of packaged functionality, would go through all phases. Projects that fall into an existing known domain but have completely new features would skip discovery and start with the analysis phase. Projects that are updates or expansions in the scope of existing features in an existing domain would jump right to the innovation phase.

4. Agile Software Project Discovery

As discussed above, projects that are within new domain start with a discovery process to define the scope and nature of the project. The discovery process covers vision statement, feature hierarchy, ball park investment, and project data sheet.

4.1. Vision Statement

The idea with the vision statement is to provide a concise overview of the project from a customer perspective. The vision statement should pass the "elevator test" and communicate to the target customer key benefits and the return on investment. (Highsmith 2004) The vision statement should be developed collaboratively with customers, managers, and developers to make sure that the vision is understood and defined by all participants in the project. It is also important to recognize software projects do not happen in a vacuum and include constraints in the vision. For example, if the software project must run or work within a certain project architecture, this should be included in the vision. The key element is that the vision will set the scope and foundation for the entire project. It is under this vision that the project will change and evolve into a finished product.

4.2. Feature Hierarchy

All software projects at some level breakdown into sets of functionality or features sets that the user/customer would find valuable. (Cohn 2004) These features are expansions on the product visions and should represent the functionality at a high level. The key here is not to define/document the functionality, but to capture the essence of the user/customer vision at a feature specific level. It addition to capturing the features to be provided, it is also important to understand relationships between features at a high level. A good high level model for understanding these relationships is to put features into a hierarchy and then notate any interdependence between features. This hierarchy, in turn,

provides a tool for breaking a project into feature modules, development iterations, feature teams, and prioritizing development.

4.3. Ballpark Investment

One of the more difficult aspects of any software project discovery process is ball parking the investment in terms of physical and human resources. This investment ballpark is typically compared to return to determine if the project is accepted or rejected, so it is a critical piece. The problem is, how does a developer ballpark a project at a point where nothing has been defined beyond high level features? The key to estimating ballparks for a project is it takes experience and practice. Through development experience and practice, a developer learns how to examine a product feature hierarchy and then extrapolate a ballpark person month estimate for each feature set within the hierarchy.

Software development is an uncertain process that evolves as it proceeds, which does not lend itself to precise production and schedules. For this reason, especially during the discovery stage, estimates should be limited to ballparks and not be considered precise commitments. One approach to mitigate risks that estimates are off is to have two developers estimate each of the features independently. Then uncertainty buffer is calculated as two standard deviations by taking the square root of the sum of squares of difference between each feature estimate.

$$2\sigma = \sqrt{(F_1E_1 - F_1E_2)^2 + (F_2E_1 - F_2E_1)^2}$$

Figure 1 Uncertainty buffer calculation (Cohn 2006)

4.4. Project Data Sheet

The project data sheet provides a one sheet overview of entire the project that covers vision statement, customers, feature hierarchy, issues/risks, ballpark estimate (days effort or person months), trade off matrix, client benefits, performance attributes, and stakeholders. The key to the document is that it is written in business language that explains the return on investment and how the project delivers on the vision.

5. Agile Software Project Analysis

Once the domain is known, a project can move into the analysis phase to document and model the software project. This consists of developing use case lists, models, interface mockups, innovative estimates, and assessment documents.

5.1. Use Case List

The first step to software project analysis is to define the use case list for all features. Each use case should contain a title and a brief one paragraph synopsis. This synopsis should describe the business requirements or main success scenario. (Cockburn 2002)

5.2. Modeling

Modeling is a logical design phase that provides models to conceptualize how features would work both standalone and as a system. The conceptual models inherently should be abstract and not depict technical design or implementation specific details. These conceptual models are implemented as activity and entity relationship diagrams.

5.2.1. Activity Diagram

An activity diagram models the steps or activities of a use case. They are similar to flow charts in that they graphically depict the flow of activities. However, they are different in that they provide a mechanism to depict activities that occur in parallel. (Bentley 2007)

5.2.2. Entity Relationship Diagram

Activity diagrams are typically focused on a specific use case or groups of use cases. The entity relationship diagram is a step back to look at the system as a whole and how the data entities that underlay use cases will be related and reused between use cases.

5.2.3. User Interface Mockups

User Interface mockups are visual models of the screens or pages that the user would use to interact with the system. These visual models are tied back to activity diagram and are shown in sequence in relation to a specific user activity.

5.3. Innovation Estimate

For software development projects, the innovation estimate comes down to human resources in terms of a day's effort. The day's effort estimate starts with a bottom up estimate for each use case assuming an ideal day where the developer is fully focused and committed. Each use case should be considered as a mini project, and the ideal day estimate should encompass time for design, programming, testing, and implementing feedback maintenance. The ideal day estimate multiplied times the daily resource cost will provide the investment estimate. It is important to note that the ideal day estimate

does not give an actual schedule. The actual schedule would depend on the resources, timing for client feedback, dependencies between features, and a host of other constraints that would fall out during the innovation phase.

5.4. Assessment Document

The assessment document expands on the vision and feature hierarchy defined in the discovery phase and covers use case lists, activity diagrams, relational entity diagrams, interface mockups, and ballpark estimate. It should also tell the reader stakeholders, performance attributes, issues/risks, and how design delivers on vision.

6. Agile Software Project Innovation

Once the domain is known and the specific features to be implemented have been documented and modeled the project is ready for the innovation phase. The innovation phase consists of iteration planning, iteration work, and feedback.

6.1. Iteration Plan

The iteration plan is an approach used in agile software development to break a project down into mini projects that cover sets of use cases in each iteration. The overall idea with this approach is to turn development into a collaborative process with the customer and illicit feedback early and often. This approach, in turn, allows development to adapt and evolve with change.

The iteration plan should break down into two-four week iterations and can vary within the plan itself. Exact iteration length would vary depending on use cases, customer

availability, and resource availability. It is also recommended that high priority or high risk use cases be placed in early iterations. This allows high impact iterations to be evolved into design at the most cost effective point.

To place the iteration plan onto a schedule, the developer first converts the ideal days to real days that represent true resource needs. Also to develop a schedule the developer must understand the concept of velocity. Velocity is a measure of a team's rate of progress. (Cohn 2004) It is found by calculated the number of real days of work that a team completes per iteration. For example, if the development team can commit 60% of available time to a project on ten day iterations, your velocity would be six. Development team ideal days total divided by the velocity would provide a rough framework for schedule.

6.2. Iteration Work

One of the key concepts of an iteration is to think of it as a mini project unto itself. Each iteration has a design, implementation, review, and maintenance elements. Each iteration needs to fit into the overall design of project, but it also must incorporate feedback from previous iterations and evolve with that feedback.

6.2.1. Technical Design

All design prior to iteration design is conceptual or logical. During the iteration phase, design is taken to physical level. Physical design gets into the specific technical design elements that implement use cases, for example, it will include database schemes,

objects, interfaces, and use cases. These use cases, in turn, can be used as test cases for implementation.

6.2.2. Sequence Diagram

Sequence diagrams are models for depicting use cases as messages between objects in time sequence. This approach allows visualization of parallel processes and time sequence of activities from top to bottom.

6.2.3. Use Case Diagram

Use case diagramming is a model that depicts features as use cases, actors (users), and relationships. Use cases describe the system functions from a perspective of external users and in a manner and terminology they understand. The purpose of the use case diagram is to communicate at a high level the scope of the business events that must be processed by a feature. (Bentley 2007)

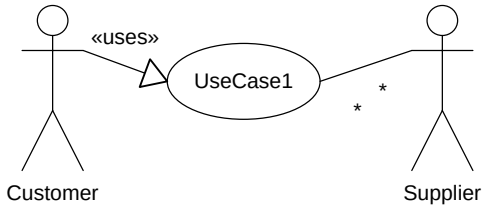

Figure 2 – Use Case Diagram Example (Bentley 2007)

6.2.4. Implement

Implementation consists of three main elements: programming, unit testing, and integration. Programming is the major element; this consists of writing code to work with

database schemes, objects, interfaces, and use case conditions as outlined in conceptual models and technical design. Programming also covers modifying existing code in a technique called refactoring to make it reusable and readable without modifying its underlying behavior. For example, if a use case condition has similar functionality to a previously implemented use case condition the developer could modify/abstract the original functionality to work in both conditions. Unit testing consists of testing specific use cases against a unit of code that provides that functionality. Integration is taking code that provides functionality under the scope of a certain use case and tying it into the software system as a whole and verifying the integration does not break the other use cases.

6.2.5. Customer Review

Reviews at the end of an iteration are conducted for two purposes. The first is to reflect, learn, and adapt from the iteration just completed. (Highsmith 2004) The second is to discuss the work to be completed for the next iteration and to apply lessons learned from previous iterations. The review should consist of development teams and customer focus groups (CFG). CFG's gathers feedback on look and feel, general operation of the software, and the use of the product in business, consumer, or operational scenarios. (Highsmith 2004).

6.2.6. Maintenance

In general incorporating CFG review feedback falls into categories of new user cases, new use case conditions, and fixes to the use cases and related conditions implemented. The first two categories go into revising the iteration plan. The last

category related to fixing use cases falls into iteration maintenance. Iteration maintenance is the last element of an iteration and consists of wrap-up programming tasks that fix or fine tune use cases scenarios.

6.3. Feedback

6.3.1. Monitoring/Metrics

One of the most common Agile development metrics is the release burn down bar chart. The release burn down shows a days' effort in backlog on the y axis and iterations on the x axis. As the back log is reduced, the y axis falls, with the added twist that changes made to scope are subtracted from y axis. For example, if the customer adds ten days effort to the second iteration, it will show bar below y axis by ten. This allows the chart to track work completed and scope changes.

6.3.2. Revise Plan

Revising the iteration plan is one of the most important steps of managing a software development project. It is during this step that the evolving/adapting aspect of agile software development is communicated to developers, customers, and managers. The first step to revising the plan is to add/remove use cases or use case conditions from backlog based on CFG review feedback. The second step is to revise velocity based on metrics shown on the project to date. The incorporation of feedback in these steps should communicate to all stakeholders' changes in effort and schedule.

7. Summary

Agile methodologies offer promising benefits in terms of quality, on target features, and costs. The challenge is applying these methodologies to the small business environment. In small business environments, budgets vary from very small to medium in scope, typically on the order of one week to three months. Projects also vary from being a completely new domain to just modifying existing features. It is not realistic to follow the same process for these widely varied scenarios. It is also important that costs associated with documenting and managing the project scale with the nature of the project.

The author, through research and experience as a small business developer with Blue Ocean Consulting, lays out an approach that breaks the Agile methodology into phases. These phases provide a framework for skipping phases of the process that are not needed in certain scenarios.

Projects that fall within a completely new domain would go into a discovery process to define vision, high level features, and ballpark of investment. Projects that fall into an existing know domain, but are completely new features would skip discovery and start with analysis phase to document and model the software project. Projects that are updates or expansions in the scope of existing features in an existing domain would jump right to the innovation phase to start the building of working code.

8. Conclusions

The Agile methodologies provide a good framework for improving software development, but they must be adjusted and modified to meet different software development environments. This paper lays out an approach for breaking down Agile methodologies, as covered within the literature review, into a structure compatible with the small business environment.

This approach breaks the overall Agile methodology into phases of discovery, analysis, and innovation. This phased approach allows aspects of Agile methodology to be skipped in scenarios where the scale of the project is small, the project is defined with limited risk of changes, or the project is a adaptation of an existing known domain.

Blue Ocean Consulting and the author has put this process into practice and found the results to confirm the value of a phased Agile approach. Clients feel the software delivered meets their expectations better and ultimately provides a much greater value than the traditional process does.

9. Recommendations for Further Research

Agile development brings a new paradigm to software development processes. The processes intersect with the team structure in terms of communication and responsibilities. For this reason, structuring the team for the Agile process would be a good area for further research. This would be an especially important research area for small businesses where team members will fill multiple roles.

Another area that could use additional research is resource planning. The Agile process inherently has a built-in mechanism to adjust the plan in terms of backlog of features to be developed, but changes to the backlog often also translate into changes in resources. Again, this is compounded in the small business environment where team members are typically working on multiple projects. A documented approach to applying Agile to resource planning would be a tremendous benefit.

10. References/Bibliography

Abrahamsson, P., Warsta, J., Siponen, M., and Ronkainen, J. 2003. New directions in agile methods: Comparative analysis. . In *Proceedings of the 25th International Conference on Software Engineering*:244-254.

Augustine, Sanjiv, Bob Payne, Fred Sencindiver, and Susan Woodcock. 2005. Agile project management: Steering from the edges. *Communications of the ACM* 48, no. 12: 85-89.

Bentley, Jeffrey Whitten Lonnie. 2007. *Systems analysis and design methods.* New York, NY: McGraw-Hill Irwin.

Cockburn, Alistar. 2002. *Agile software development*. Edited by Alistar Cockburn & Jim Highsmith. The agile software development series. Indianapolis, IN: Addison - Wesley.

Cohn, Mike. 2004. *User stories applied : For agile software development*. Addison-wesley signature series. Boston: Addison-Wesley.

_____. 2006. *Agile estimating and planning*. Edited by Robert C. martin. Upper Sadle River: Pearson Education, Inc.

Highsmith, James A. 2004. *Agile project management : Creating innovative products*. Agile software development series. Boston: Addison-Wesley.

Kent Beck, Mike Beedle, Arie van Bennekum, James Grenning, Jim Highsmith, Andrew Hunt, Robert C Martin, Steve Mellor, Ken Schwaber, Alistair Cockburn, Ron Jeffries, Jeff Sutherland, Ward Cunningham, Jon Kern, Dave Thomas, Martin Fowler, & Brian Marick. Manifesto for agile software development. http://www.agilemanifesto.org/ (accessed 9/2/2008, 2008.

Larman, Craig. 2004. *Agile and iterative development : A manager's guide*. Agile software development series. Boston: Addison-Wesley.

11. Appendix A